## Access Independent Query Definition in IBM DL/I

An implementable mapping between DL/I data bases by means of a simple conceptualization of their structure associates together the set of paths defined in various data bases and having a common query structure; that is to say, any one of the set may be used to execute a query defined on any of the others. Membership of this set is both a necessary and a sufficient condition for access equivalence with any other member of the set and allows a query system the widest possible domain of effective paths from which its access optimization routines may select the most efficient. The definition of these domains, each associated with a segment selectable for a query, may efficiently be performed once only, before online query execution. IBM's Data Language, DL/I, supports hierarchically structured data in such a way that a DL/I application is provided with a number of hierarchies, expressed as Program Communication Blocks (PCBs), against which it accesses its data. A PCB expresses both a hierarchical structure and a way of accessing that structure, namely, by accessing instances of the PCB's root sensitive segment (SENSEG) and thereafter accessing any child SENSEG within its present SENSEG. Various refinements to this scheme are available, but in essence the principle holds that PCB structure determines access sequence. Where an application is planned in advance, its optimal accessing sequence can be ascertained and appropriate PCBs generated, if necessary, along with their underlying logical data bases and indexes. When, however, a system to answer ad hoc queries on DL/I data bases is constructed, this optimization becomes more difficult. One option is to require that the user of such a system, as part of the definition of his query, specifies the PCB that should be used. This results, however, in considerable added complexity for the user and may result in highly inefficient execution of the query, for the user tends to select a PCB that gives him an intuitively acceptable view of his data rather than optimal efficiency of execution. It is desirable, however, that, despite the user quite properly taking his own view of his data, the query system should use that PCB which provides for efficient execution. Once a query has been ambiguously defined, a number of access paths, as defined in PCBs, may be appropriate for execution of the query (that is, they would all give the same result - assuming that the sequence of records comprising the result may be sorted if necessary before presentation to the user, and, for the sake of simplicity, that a single PCB is adequate for the query, but the approach is applicable to queries involving more than one PCB). A collection of PCBs having in common the property of appropriateness for a given query may be said to have a common structure, which may be termed the 'query structure'. This query structure is also possessed by the query as defined by the user, who may have defined his query on a PCB, although this is not necessary. He must have defined his query on a structure that can be mapped by the system to a query structure possessed by some PCB accessible to the query system. Any PCB is defined on either a DL/I physical data base (PDB) or a DL/I logical data base (LDB). LDBs themselves are defined on

PDBs. It is not valid to assume that PCBs are defined on LDBs and these, in turn, on PDBs (the 'special case' of a PCB directly defined on a PDB may be treated by taking the PDB to be an LDB defined on itself). The definition of a PCB on an LDB is straightforward in that the PCB has the same hierarchical structure as the LDB (or a subset of it). Identification of common query structure depends therefore on the mappings between LDBs and PDBs. Given a collection of DL/I accessible data, the PDB definitions of that data, as defined at DL/I system generation, completely specify not only all PDBs but also all LDBs that could be defined on the date. To this extent, the LDB (and PCB) definitions add nothing to the PDB definitions. What they do add is accessing ability. It should be noted that what in DL/I are termed 'logical relations' are not related to LDBs, as 'physical relations' are related to PDBs. Indeed, both 'physical' and 'logical' relations are defined in PDBs, and, for that matter, are equally physical in their realization, being implemented by pointers. The user's view of data, according to which he formulates his query, is assumed to comprise a number of hierarchies which may be termed Entry Data Bases (EDBs). An EDB may be in practice a PCB, an LDB, or such a structure of segments as might be defined as an LDB. A DL/I Physical Data Base (PDB) comprises a number of root segments of common type and their dependent segments. Each type of segment in a PDB is either the root or has a single (physical) parent segment type; thus, the (physical) structure of a PDB is determined by listing the type of the root and each other type of segment along with its parent segment type. Because of this hierarchical structure there is a one-to-one relationship between segments and paths in a PDB, where a path is an ordered set of segments, the first member of which is a root and subsequent members children of their predecessors. Thus, if segment a is a root, segment b one of its children, segment c one of b's children, then a corresponds to (a), b to (a, b), and c to (a, b, c). In addition to this physical structure, DL/I also permits the definition of logical relationships between types of segment. The 'logical parent' and 'logical child' relationships parallel their 'physical' analogs (recall that both are physically implemented). Thus, in the description above, the relation of a to b or b to c could be logical parent rather than physical parent and a (different) one- to-one correspondence would still hold. These logical relationships, however, may (but need not) hold between segments in different PDBs. If a segment, s, is physical child of a segment, p, and logical child of a segment, l, in an allowed implementation of DL/I, there may be defined a segment, r, which is physical child of l, logical child of p, and contains essentially the same information as s (not exactly the same, for certain redundant key information is copied from the physical ancestors of r into s and from the physical ancestors of s into r). The segments s and r are said to be paired . The pair relationship is one-to-one (it is assumed throughout that the DL/I data bases under consideration have consistent update rules). Other allowed implementations of logical relationships in DL/I allow neither pairing nor reference to paired segments, and reference to paired

segments without actual pairing. The latter implementation is termed virtual pairing: both s and r are defined, but r is not implemented. A reference to r results in the equivalent access to s and processing that simulates, for an application program, what would have resulted had r been implemented (by 'physical' pairing, as described above). It may be assumed that every segment that is a logical child has a paired segment. The paired segment name will denote one of the following Œa physical pair Œ a virtual pair Œ the same segment Œ nothing (in the case of a 'unidirectional' logical relationship) In the last case DL/I protects against any attempt to use the supposed paired segment because no PCB can be defined that would refer to it. A DL/I application program accesses a segment that is a logical child through a Logical Data Base (LDB). A LDB, as noted above, is not additional to, but is defined on one or more PDBs. Starting from the root segment of the LDB, which is defined (or sourced) on the root segment of a PDB, such that the parentage relationships in each LDB mirror those in the PDB, until a LDB segment would be sourced on a PDB segment that is a logical child. At this point a concatenated segment is defined in the LDB. It has two sources, the logical child segment and its logical parent (which are, in appearance to any application using the LDB, appended). A child segment of a concatenated segment (in the LDB) may be sourced on any of Œa physical child of the first source segment Œ a physical child of the second source segment Œ the physical parent of the second source segment In the last case, a child segment of that segment may also be sourced on the physical parent of its source, and so on until the source is the root of a PDB. Apart from this inversion and the sourcing of concatenated segments, the source of a dependent segment in a LDB is the physical child of the source of that segment's parent in the LDB. In view of the inversions allowed in LDBs it is clear that the sources of the segments in a LDB do not parallel, in their physical or logical relationships, the structure of the LDB. That is to say, if a and b are the sources of two LDB segments that are respectively parent and child, a may be physical child of b. Consequently, there is not a one-to-one relationship from the sources of LDB segments to the paths to those segments, a one-to-many relationship being the general case. An instance of some type of physical segment which is the source of a LDB segment may correspond to no path, to one path, or to many paths in the LDB. In effect, an LDB segment is not identifiable with its source or sources. It may be identified with (or defined as an object bearing a one-to-one relationship with) the path to itself as defined in the LDB, or the path comprising the sources of those segments in the LDB path. Thus, source paths, say, (a), (a, b, c) and (a, b, c, d), might correspond to LDB segments A (the root, sourced on a), B (a concatenated segment, sourced on b and c), and D (sourced on d). If the access sequence of the LDB is imposed, these paths of sources are read. But if access sequence is not imposed, and this is precisely the objective in a query system that attempts to optimize access efficiency, some other mechanism must be imposed to ensure reading either of these paths, or of another set of paths

in one-to-one relationship to them and containing the same information. In any event, the LDB-like structure of an Entry Data Base may be exploited in assisting the user's query definition, for this is formulated on exactly the required hierarchical view, despite that view not being bound to the actual access sequence. A second consideration with LDBs is that, taking the segments p, s, r, and l, as described above, there could be two LDBs, as follows: ŒLDB 1 root sourced on p and dependent segment s and l Œ LDB 2 root sourced on l and dependent segment on r and p It will be seen that LDBs 1 and 2 comprise the same information, and either may be used (by means of a PCB) to process a query defined on the other. It is therefore, although sufficient, not necessary to identify query structures between two LDBs (and therefore PCBs) that their PDB source paths be identifiable. However there is a certain conceptualization of DL/I data bases that does define such identity. It should be noted that the mere identification of segments that are paired with each other, or, what is equivalent, the identification of query structures when they differ only in that one contains a segment where the other contains its pair, also fails. Such failure occurs when a segment has the same type of segment as both physical and logical parent and therefore identification of that segment with its pair fails to distinguish the relation of the physical to the logical parent from its converse. Thus, identity of source paths is a sufficient, but not necessary, condition. Given identification of logical child segments with their pairs, it becomes a necessary, but not a sufficient, condition. In DL/I the concepts 'physical parent', 'logical parent', and 'pair' are all primitive (that is, formally undefined). If however, as the present arrangement assumes, each logical relationship involves segment pairing, then the 'logical parent' relationship is definable as 'parent' of this pair. On each PDB a Conceptual Data Base (CDB) is defined, having one segment defined, or founded, on each PDB segment, and having the same structure of parentage. In addition, if any PDB segment, s, is the logical parent of a segment, but that segment's pair is not defined in the PCB as physical child of s, then a segment is defined in the CDB to be (physical) child of the segment founded on s and pair of the segment founded on the (unpaired) logical child of s. It may be noted that the terms 'parent' and 'child', when pertaining to CDB segments, are always 'physical'. Any path in an LDB may now be mapped onto a path in one or more CDBs. Each non-concatenated segment is mapped onto the CDB segment founded on the source of the LDB segment. Each concatenated segment is mapped onto a sequence of three CDB segments:

1. The CDB segment is founded on the first source of the concatenated segment, if that source is the physical child of the source of the LDB parent of the concatenated segment; otherwise, the CDB segment that is pair of the CDB segment is founded on the first source of the concatenated segment; 2. the CDB pair of 1; and
3. the CDB parent of 2 (founded on the second source of the concatenated segment). In this arrangement, Œ If two paths map to the same sequence of CDB

segments, then they have the same query structure. Œ If the sequence of CDB segments to which one path
maps is the exact inverse of the sequence to which
another path maps (that is, has the same segments but
in reverse order), then the two paths have the same
query structure. Œ If a pair of paths, starting from a common
segment, are each mapped onto a sequence of CDB
segments, and one of those sequences inverted and
joined to the other at the common segment, then, if the
resultant sequence is that mapped from a further path,
then the pair of paths has the same query structure as
that further path. In sum, a sequence of CDB segments constitutes a query structure, irrespective of the point of entry for accessing purposes, whether at either end or in the middle. It may be noted that the last rule of common query structure, above, means that a query defined by the user as two paths with a common root may be executed using either a single path or two paths with a common (but not necessarily the same) root. If a query system is, in advance of online execution, provided with definitions of the DL/I data bases and PCBs to be used by it and of the segment structures of the EDBs upon which users will formulate queries, it can determine for each choice of EDB segment by a user: 1. the path in the EDB associated with that segment; 2. the path of PDB segments that is the source of 1; 3. the path of CDB segments founded on 2; 4. each path of sensitive segments in each available
PCB; 5. the LDB path defined by 4; 6. the path of PDB segments that is the source of 5; 7. the path of CDB segments founded on 6; 8. those CDB paths (7) that are identical to that (3)
of the potential query; and, therefore, 9. those paths of sensitive segments (4) that have
the query structure necessary to access the CDB
segment.

This establishment of correspondence having been performed before online execution, it remains to combine paths of sensitive segments (when a query involves two segments not in a common path) and to select the most efficient path for execution of the query. Neither of these tasks can (practically) be performed until an actual query is formulated. Although not all access determination can be performed before online execution, the described arrangement, if implemented as an offline process, does significantly reduce the workload (and therefore improve the response of) the online system.